A notebook and a pen next to a keyboard

Description automatically generated

# **Industrial Control Systems Security**

# **Project**

Team algorithm "SHA-512 (64 bytes)"

Professor: Ing. Alexandru Costache

Group: 4MF431

Master Studies, CyberSecurity, *Transilvania* University of Braşov.

Students:

Daniel CALAVER

Dragos VÂJU

Norbert WILLMAN

Year **2023 / 2024**

CONTENS

|  |  |  |
| --- | --- | --- |
| 1. Contents |  | 3 |
| 1. Project Overview |  | 4 |
| 1. Project Specifications |  | 5 |
| 1. Software Architecture |  | 6 |
|  | Button Driver Module | 6 |
|  | LED Driver Module | 7 |
|  | Cybersecurity Driver Module | 7 |
|  | Application Module | 8 |
| 1. SHA-512 Algorithm |  | 9 |
|  | SHA Family | 9 |
|  | SHA-512 | 11 |
| 5. Bibliography |  | 14 |

1. **PROJECT OVERVIEW**

The project implements and integrates a module that calculates a hash using the SHA 256 algorithm overt the button driver (first 2kB of the driver, memory range 0x2000 – 0x27FF). The module assures the security over the system already implemented during the semester classes.

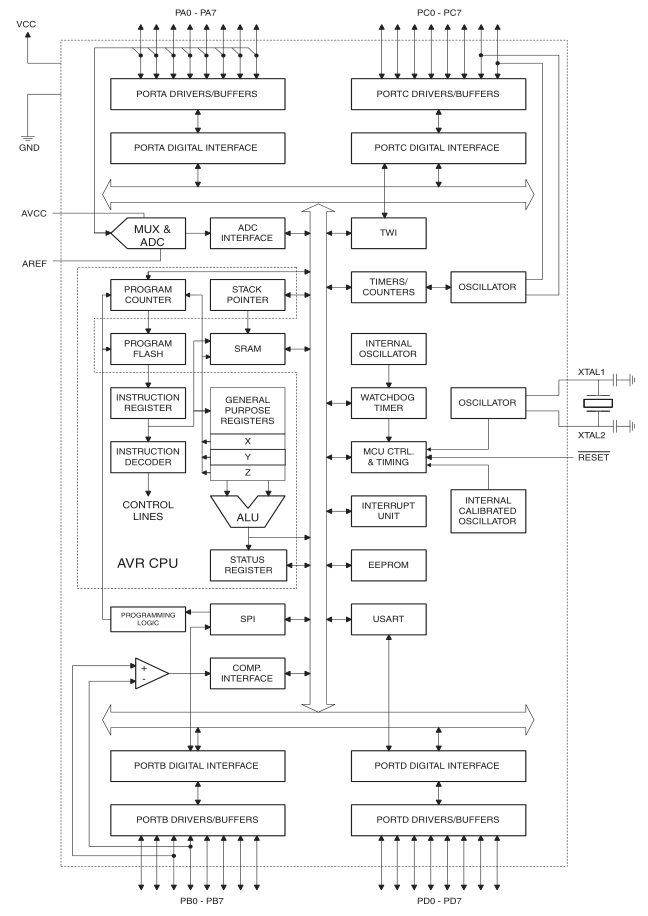
The protection mechanism works as follows: in case the protected driver gets modified, the system will detect the change and protect itself by closing main functionality of the system. Once the security module detects a change in between the pre-calculated hash and the one calculated during runtime, the button detection and led toggling will stop working.

The system performs (trigger) a hash calculation over the protected area every 10 seconds.

The MicroController

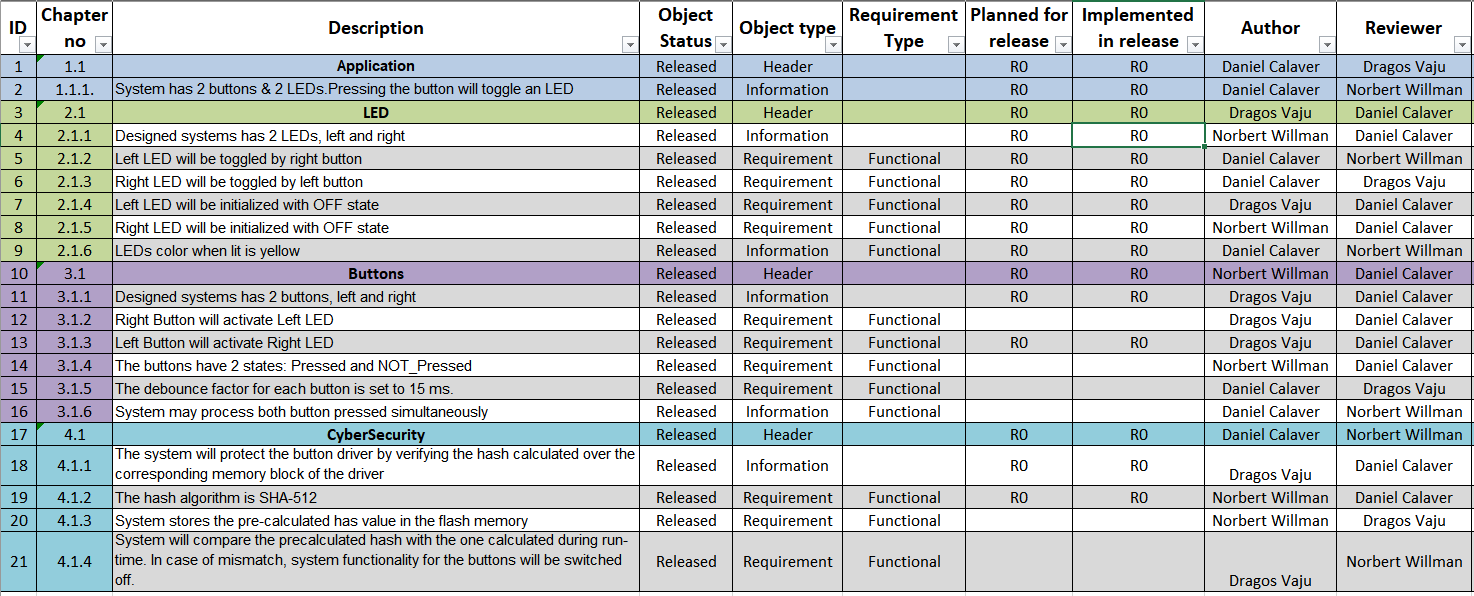
The project is implemented in Proteus simulating an Atmel®AVR®ATmega32 microcontroller.

The Atmel ® AVR ® ATmega32 is a low-power CMOS 8-bit microcontroller based on the AVRenhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega32 achieves throughputs approaching 1 MIPS per MHz allowing the system designer tooptimize power consumption versus processing speed.



**Figure 1** – Block diagram of ATmega32

1. **PROJECT SPECIFICATIONS**

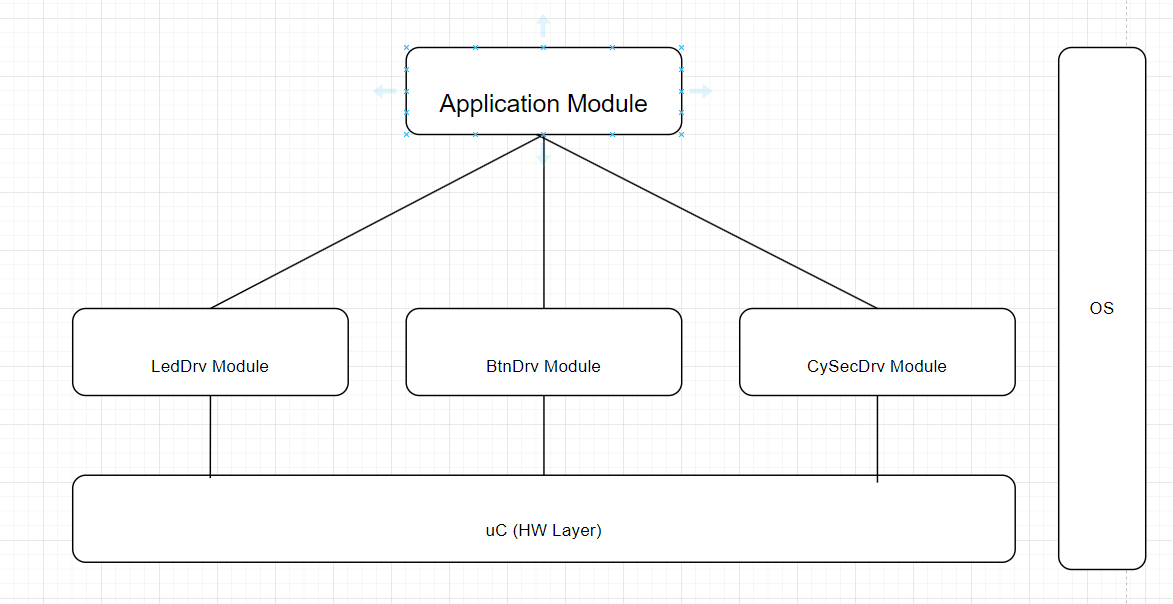


**3 . SOFTWARE ARCHITECTURE**

The project’s architecture consists of 4 main modules:

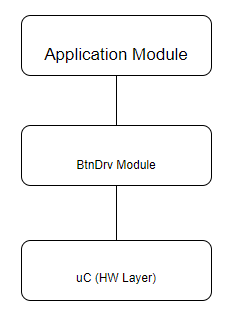
* Button Driver Module
* LED Driver Module
* Cybersecurity Driver Module
* Application Module

Modules’ interaction one to another is visually presented in Figure 2, below:



**Figure 2** – Software Architecture

* 1. **Button Driver Module**

The purpose of this driver is to read and detect a state of the buttons from the microcontroller. It will communicate this status to the Application Module. This driver is configurable and it can support up to eight buttons.

The following functions are implemented:

* *void BtnDrvInit()* ==> Funtion to provide initialization for the module. It will set the states of the button to the initialized state based on one reading
* *void BtnDrvMain()* ==> The main function of the driver. It will be called every 10ms, via a task.
* *teBtnDrvState BtnDrvGetButtonState (teBtnDrvID)* ==> Function will provide the logical state of the button, returns the stable state of the button

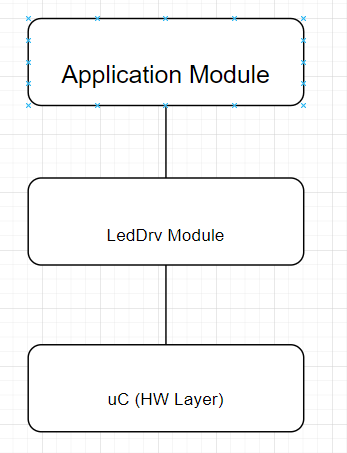
The following variables / Type definitions are used:

* *teBtnDrvID (enum) ==> BTN\_LEFT, BTN\_RIGHT, MAX\_NO\_BTN*
* *teBtnDrvState (enum) ==> PRESSED, NOT\_PRESSED*

**Figure 3** – Button Driver Module

* 1. **LED Driver Module**

**Figure 4** – LED Driver Module



The purpose of this module is to set the LED's state. The LED driver can control up to 8 LEDs based on configuration. The LEDs can be turned ON, OFF or toggled.

The following functions are implemented:

* *void.LedDrvInit() ==>* Function to provide initialization for the module. It will set the initial state of the LED driver variables.
* *void.LedDrvToggleLedState(teLedDrvID eLedID) ==>*  will toggle the LED based on parameter
* *void.LedDrvSetLedState(teLedDrvID eLedID, teLedDrvState eLedState) ==>* will set a certain state (ON/OFF) of the LED based on paramenters

The following variables / Type definitions are used:

* *teLedDrvID (enum) ==> LED\_LEFT, LED\_RIGHT, MAX\_NO\_LEDS*
* *teLedDrvState (enum) ==> ON, OFF*
  1. **Cybersecurity Driver Module**

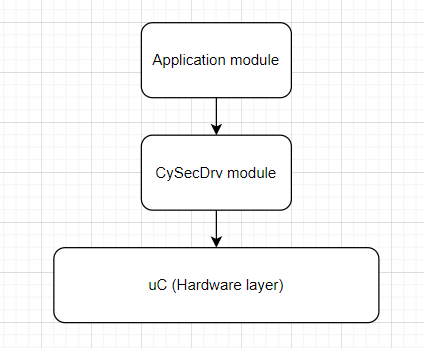
The CySecDrv module will calculate the corresponding hashes over memory blocks. It will also set the system security stare based on the calculated hash value.

The following functions are implemented:

* *void CySecDrvInit() ==> Initialize CySecDrv and calculates first secure state*
* *teCySecDrvStatus CySecDrvGetSecurityState () ==> returns the current security state of the system*
* *void CySecDrvMain() ==> Function to provide initialization for the module.*

The following variables / Type definitions are used:

*teCySecDrvStatus (enum) ==> SECURED, NOT\_SECURED, UNDEFINEDtcButtonHash --> will calculate the hash for the button memory block*



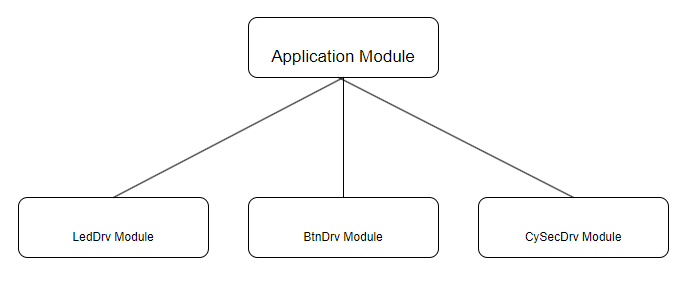
**Figure 5** – Cybersecurity Driver Module

* 1. **Application Module**

The purpose of this module is to create the connections between all other modules.

The following functions are implemented:

* *void ApplicationInit()* --> The function provides initialization the Application module and its dependencies
* *void ApplicationMain() -->* Application main function



**Figure 6** – Application Module

**4. SHA-512 ALGORITHM**

**4.1 SHA Family of Algorithms**

SHA is the acronym for Secure Hash Algorithms. These are a family of cryptographic hash functions, published by NIST – the US - National Institute of Standards and Technology as a U.S. Federal Information Processing Standard (FIPS).[1]

* [**SHA-0**](https://en.wikipedia.org/wiki/SHA-0): A [retronym](https://en.wikipedia.org/wiki/Retronym) applied to the original version of the 160-bit hash function published in 1993 under the name "SHA". It was withdrawn shortly after publication due to an undisclosed "significant flaw" and replaced by the slightly revised version SHA-1. [1]
* [**SHA-1**](https://en.wikipedia.org/wiki/SHA-1): A 160-bit hash function which resembles the earlier [MD5](https://en.wikipedia.org/wiki/MD5) algorithm. This was designed by the [National Security Agency](https://en.wikipedia.org/wiki/National_Security_Agency) (NSA) to be part of the [Digital Signature Algorithm](https://en.wikipedia.org/wiki/Digital_Signature_Algorithm). Cryptographic weaknesses were discovered in SHA-1, and the standard was no longer approved for most cryptographic uses after 2010. [1]
* [**SHA-2**](https://en.wikipedia.org/wiki/SHA-2): A family of two similar hash functions, with different block sizes, known as *SHA-256* and *SHA-512*. They differ in the word size; SHA-256 uses 32-bit words where SHA-512 uses 64-bit words. There are also truncated versions of each standard, known as *SHA-224*, *SHA-384*, *SHA-512/224* and *SHA-512/256*. These were also designed by the NSA. [1]
* [**SHA-3**](https://en.wikipedia.org/wiki/SHA-3): A hash function formerly called [*Keccak*](https://en.wikipedia.org/wiki/SHA-3), chosen in 2012 after a public competition among non-NSA designers. It supports the same hash lengths as SHA-2, and its internal structure differs significantly from the rest of the SHA family. [1]

SHA-2 were first published in 2001 by the United States National Security Agency (NSA), has released the patent under a royalty-free license.

The SHA-2 family consists of six hash functions with digests (hash values) that are 224, 256, 384 or 512 bits: **SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256**. SHA-512 is a novel hash function computed with eight 64-bit words.[2]

**4.2 SHA-512**

SHA-512 is just one of several algorithms in the Secure Hashing Algorithm (SHA) family.

SHA-512 function has:

* Final output size (bits): 512-bit message digest
* Internal state size (bits): 512 (8 words × 64 bit each)
* Block size (bits): 1024 or a sequence of sixteen 64-bit words.
* Requires compute rounds per block: 80
* Operations performed in calculation: And, Xor, Or, Rot, Shr, Add (mod 264)
* Security against collision attacks\* (bits): 256
* Preimage resistance\*\* for 57 out of 80 rounds of SHA-512

\* a **collision attack** on a cryptographic hash tries to find two inputs producing the same hash value [4]

\*\* a **preimage attack** on cryptographic hash functions tries to find a message that has a specific hash value.[3]

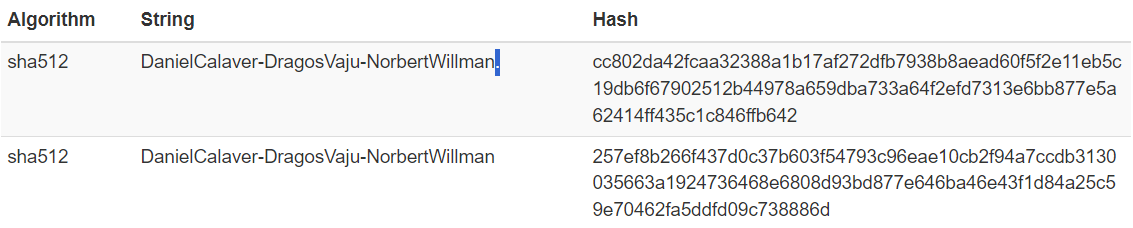
Advantages of SHA-512 are:

* **Longer Hash Length**: SHA-512 produces a 512-bit hash, which is longer than many other hash functions
* **Lower Collision Chance**: The larger the hash, the lower the chance of collisions (that is, different inputs producing the same hash). Collisions are a significant issue in hash functions as they can lead to confusion and security issues
* **Data Integrity and Authenticity**: SHA-512 is widely used in a variety of security applications and protocols, including TLS and SSL, PGP, SSH, IPsec, and more. It ensures data integrity and authenticity.
* **Sensitive to Changes:** If even a single character in your document changes, SHA-512 will produce a completely different hash.

Two significant disadvantages of SHA-512 are:

* **Memory Usage:** SHA-512 uses more memory than SHA-256 or SHA-1. This is a limiting factor in resource-constrained environments or embedded systems.
* **Compute speed:** SHA-512’s performance is not as efficient as SHA-256. It’s slower because it has to process the larger 64-bit data chunks in parts. Computing SHA-512 hashes over extremely long messages is very slow.

Example of how different the obtained hash value is for a text where a character (.) is added. There is a total of 253 different bits between the 2 hashes provided.

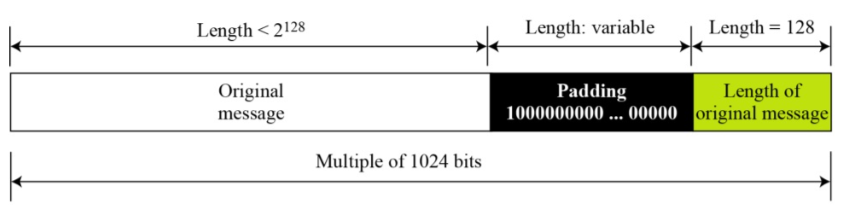


**Figure 7** – Adding a dot at the end of a string, changed the calculated hash significantly. Hashes obtained via <https://md5calc.com/>

**4.3 SHA-512 Algorithm Steps**

PreProcessing - Input Formatting in SHA-512

In the SHA-512 hash function, the input formatting phase involves preparing the message for processing. The algorithm accepts messages of varying lengths and employs a padding scheme to standardize input sizes. Specifically, the message is padded with a series of bits to ensure it meets the necessary block size requirements for the algorithm. This padding includes the addition of a '1' bit followed by a series of '0' bits and a representation of the original message length. The input formatting phase ensures uniformity in message length, enabling SHA-512 to process messages of any size consistently.



**Figure 8** – Visually explaining Pre-Processing in SHA-512

4.3.1 Padding the Message

The algorithm starts by appending a bit 1 to the end of the message. This is followed by appending k bits 0, where k is the smallest, non-negative solution to the equation l + 1 + k ≡ 896 mod 1024 (l being the length of the original message in bits). Finally, a 128-bit big-endian representation of l is appended to the message. This process ensures that the total length of the message is a multiple of 1024 bits, the block size for SHA-512. [5]

4.3.2 Parsing the Message

Message is divided into 1024-bit blocks. These blocks will be later processed one by one.

Since the 1024 bits of the input block may be expressed as sixteen 64-bit words, the first 64 bits of message block are denoted M1 , the next 64 bits are M2 , and so on up to M15. [6]

Hash Buffer Initialization in SHA-512

Hash buffer initialization is a critical step in the SHA-512 algorithm that establishes the initial state of the hash function. The algorithm employs an 8-word hash buffer (A, B, C, D, E, F, G, H), initialized with specific constant values derived from the square roots of prime numbers. These initial values create a starting point for the iterative processing that follows. The hash buffer serves as a repository for intermediate hash values during the computation, evolving through multiple rounds of processing to ultimately produce the final hash output. Proper initialization is essential for the algorithm's security and the creation of a unique hash value.[5]

4.3.3 Setting the Initial Hash Value

These words were obtained by taking the first sixty-four bits of the fractional parts of the square roots of the first eight prime numbers.[6]

a = H0 = 6a09e667f3bcc908

b = H1 = bb67ae8584caa73b

c = H2 = 3c6ef372fe94f82b

d = H3 = a54ff53a5f1d36f1

e = H4 = 510e527fade682d1

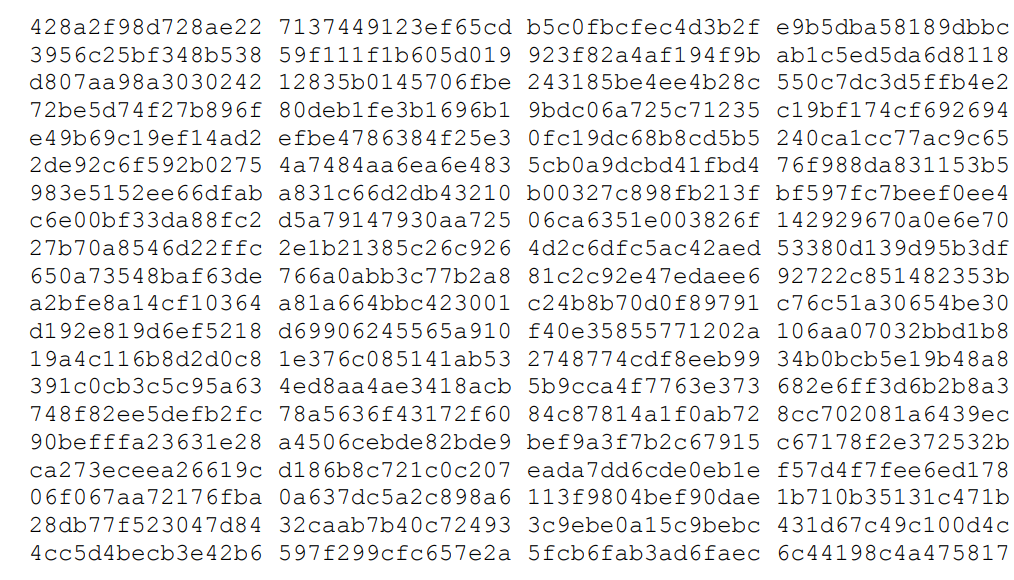
f = H5 = 9b05688c2b3e6c1f

g = H6 = 1f83d9abfb41bd6b

h = H7 = 5be0cd19137e2179

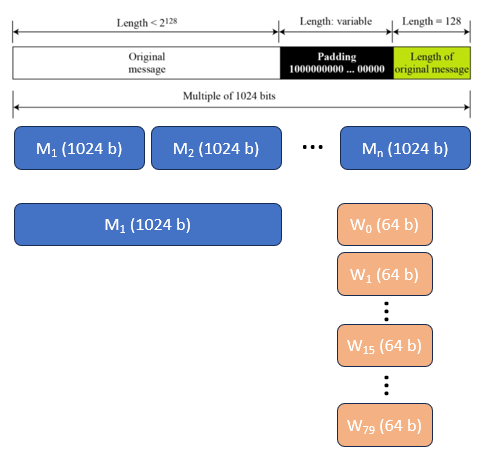
Define the constants.

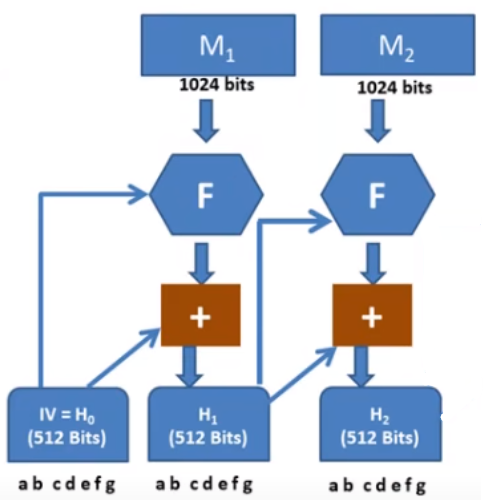
SHA-512 use the same sequence of eighty constant 64-bit words, K0, K1 … K79. These words represent the first sixty-four bits of the fractional parts of the cube roots of the first eighty prime numbers. In hex, these constant words are (from left to right) [6]



Message Processing in SHA-512

The message processing phase in SHA-512 involves dividing the padded input message into blocks and iteratively processing each block. Each block undergoes a series of complex operations, 80 steps, including bitwise operations, logical functions, and modular addition, in multiple rounds. The input block is combined with the current state of the hash buffer, and the result becomes the new state for the next iteration. **This iterative process ensures a high degree of diffusion and avalanche effects, making it computationally infeasible to reverse engineer the original input from the hash output.** [5]





**Figure 9** – Message processing and Message Digest in SHA-512 [7]

H3: Generating the Output in SHA-512

The final phase in SHA-512 involves generating the hash output from the evolved state of the hash buffer. The 512-bit hash value is constructed by concatenating the values of the eight words in the hash buffer, creating a fixed-size representation of the processed input message. The resulting hash output exhibits properties such as collision resistance and unpredictability, forming the basis for its cryptographic strength. **This final step in the SHA-512 algorithm produces a hash value that is unique to the input message, providing a secure and reliable means of verifying data integrity in various cryptographic applications.** [5]

**5. BIBILIOGRAPHY**

[1] <https://en.wikipedia.org/wiki/Secure_Hash_Algorithms> Wikipedia - Accessed: 22.05.2024.

[2] <https://en.wikipedia.org/wiki/SHA-2> Wikipedia - Accessed: 22.05.2024.

[3] <https://en.wikipedia.org/wiki/Preimage_attack> Wikipedia - Accessed: 22.05.2024.

[4] <https://en.wikipedia.org/wiki/Collision_attack> Wikipedia - Accessed: 22.05.2024.

[5] <https://debugpointer.com/security/sha512-overview> - Accessed: 23.05.2024.

[6] FIPS PUB 180-4s publication: <http://dx.doi.org/10.6028/NIST.FIPS.180-4>

[7] SHA 512 - Secure Hash Algorithm - Step by Step Explanation – Satish C J – Youtube – Accessed 01.05.2024 <https://www.youtube.com/watch?v=JViXozmJnSk>

[8] Breaking Down: SHA-512 Algorithm, Aditya Anand <https://infosecwriteups.com/breaking-down-sha-512-algorithm-1fdb9cc9413a> – Accessed 01.05.2024